Introduction

- Chapter 3: Controllers
  - Control input/output: single bit (or just a few) representing event or state
  - Finite-state machine describes behavior; implemented as state register and combinational logic

- Chapter 4: Datapath components
  - Data input/output: Multiple bits collectively representing single entity
  - Datapath components included registers, adders, ALU, comparators, register files, etc.

- This chapter: custom processors
  - Processor: Controller and datapath components working together to implement an algorithm

RTL Design Method

- Chapter 2: Combinational Logic Design
  - First step: Capture behavior (using equation or truth table)
  - Remaining steps: Convert to circuit

- Chapter 3: Sequential Logic Design
  - First step: Capture behavior (using FSM)
  - Remaining steps: Convert to circuit

- RTL Design (the method for creating custom processors)
  - First step: Capture behavior (using high-level state machine, to be introduced)
  - Remaining steps: Convert to circuit

RTL Design Method: “Preview” Example

- Soda dispenser
  - c: bit input, 1 when coin deposited
  - a: 8-bit input having value of deposited coin
  - s: 8-bit input having cost of a soda
  - d: bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda

How can we precisely describe this processor’s behavior?
Preview Example: Step 2 -- Create Datapath

- Need tot register
- Need 8-bit comparator to compare s and a
- Need 8-bit adder to perform \( \text{tot} = \text{tot} + a \)
- Wire the components as needed for above
- Create control input/outputs, give them names

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (bit), a (8 bits), s (8 bits)</td>
<td>d (bit)</td>
</tr>
<tr>
<td>Local registers: tot (8 bits)</td>
<td></td>
</tr>
</tbody>
</table>

Wait  | Add  | Disp  | Init
|-------|------|-------|------
| \( d = 0 \) | \( \text{tot} = \text{tot} + a \) | \( d = 0 \) | \( \text{tot} = 0 \) |

Controller Datapath

Controller's inputs
- External input \( c \) (coin detected)
- Input from datapath comparator's output, which we named \( \text{tot} < s \)

Controller's outputs
- External output \( d \) (dispense soda)
- Outputs to datapath to load and clear the tot register

Design Challenge

- Create a high-level state machine that describes the following system behavior.
  - The system has an 8-bit input \( A \), a single-bit input \( d \), and a 32-bit output S.
  - On every clock cycle, if \( d = 1 \), the system should add \( A \) to a running sum and output that sum on S.
  - If \( d = 0 \), the system should instead subtract. Ignore issues of overflow and underflow. Don’t forget to include an initialization state.
  - Hint: declare and use an internal register to keep the sum. (b) Add a 1-bit input \( \text{rst} \) to the system. When \( \text{rst} = 1 \), the system should clear its sum back to 0.

Due:
- Next Lecture (Friday, October 17)
- Extra Credit (Homework)
  - 2 points